REMARKS

Claims 1, 9, 12, 18, 23, 30, 31 and 37 are amended. Claims 1-42 remain in the application for consideration. In view of the following amendments and remarks, Applicant respectfully solicits allowance of the application and furtherance onto issuance.

Specification Objection

The Specification has been objected to for informalities pertaining to missing Serial Numbers of various patent applications that are related to the present application, and which are listed in the "Related Applications" section appearing on page 1 of the present application. The Specification has been amended to include the specific Serial Numbers of the applications that are referenced in the "Related Applications" section of the application.

Claim Objections

Claims 23, 30, 31 and 37 are objected to because of various informalities. Claims 23 and 37 have been cosmetically amended to address the informalities indicated by the Office.

Additionally, claims 30 and 31 have been cosmetically amended to address the informalities. Applicant sincerely appreciates the Examiner's attention to detail.

§ 101 Rejections

Claims 1-7, 9-10, 12-16, and 18-24 stand rejected under 35 U.S.C. § 101 as being directed to non-statutory subject matter. Specifically, the Office has argued that the claimed subject matter is directed to non-functional descriptive material.

Applicant has amended the claims to recite that the software architecture is ***embodied on a computer-readable medium***. Accordingly, as such, these claims and their dependent claims recite statutory subject matter.

§ 102 and § 103 Rejections

Claims 1, 8-9, 11, 18, 25, 40 and 42 stand rejected under 35 U.S.C § 102(b), as being anticipated by U.S. Patent No. 5,764,985 to Smale.

Claims 2-7, 10, 12-17, 19-24, 26-39, and 41 stand rejected under 35 U.S.C. § 103(a) as being obvious over Smale in view of U.S. Patent No. 6,421,656 to Cheng.

Before discussing the substance of the Office's rejection of the above-mentioned claims, the following discussion of the references to Smale, Cheng and the Applicant's disclosure is provided to assist the Office in appreciating the patentable distinctions between the Applicant's claimed embodiments and the cited references.

The Smale Reference

Smale's disclosure is directed, generally, to methods and systems of coordinating software extensions by utilizing a central management system. As described in Smale's "Background" section, in conventional computer operating systems, application programs have common access to a number of system-provided software routines, such as service providers that provide low-level interfaces to hardware devices. Software developers often find it desirable to extend the functionality of these commonly-accessed routines, without modifying the routines themselves, by effectively adding additional software code thereto. At present, such extensibility is accomplished by loading the additional software code, known as an extension or a monitor, into the system memory and then employing a scheme to pass system calls placed by application programs to the extension instead of passing the calls to the system-provided routine. The extension thereby obtains control of the system calls.

The main problem with the above schemes, as noted by Smale, is that each extension is essentially independent of other such extensions. As a result, the operation of an extension is often negatively impacted by the operations of other extensions which it knows nothing about. For example, problems frequently arise because of the order that the extensions are loaded into the system, such as when a second, subsequently loaded extension mistakenly fails function calls that are intended to be handled by a first, previously loaded extension.

Further, as noted by Smale, extensions must also be written to understand many of the behaviors of the function being extended in order to properly deal with various types of calls thereto. Moreover, extensions are unable to pass calls to extensions loaded after them, thereby reducing the flexibility and usefulness of extensions. As a result, it is often necessary to alter an existing extension rather than adding additional extensions in order to further modify the behavior of a called function.

In addition, as noted by Smale, when employing the above-described address-substitution scheme, the extensions must independently evaluate each function call to determine whether it wants to handle the call or pass it on to the interrupt handler. As a result, each extension possesses a substantial amount of repetitive code for determining if each function call passed thereto is within its area of interest.

To address this and other problems as described by Smale, Smale discloses a system that is best appreciated, preliminarily, with reference to Smale's Fig. 1.

There, a layered software architecture is shown and comprises application programs 23, 24 that can initiate function calls via an application program interface (API) 25 to request the performance of various operations. From the API 25, the call is received by a routing component 26, which routes the requested function call through a service provider interface (SPI) 27 to one or more appropriate service providers. One of the service providers, e.g. service provider 28, may handle the request directly, or alternatively may call on other, lower-level service providers (not shown). Another service provider 29 is provided to call on operating system functions when so required by the requesting function call.

Figs. 2A and 3 show a routing component 26 has been modified to include a notification manager 32 in accordance with Smale's invention. The notification manager 32 provides a centralized subsystem that notifies ***registered extensions*** 34, 35 of each function call both before (pre-notification) and after (post-notification) the call is made to the service provider.

Fig. 2A illustrates the general flow of Smale's operation when a typical function call is made from an application program. With a layered architecture, a request router 30 of the routing component 26 initially receives the requesting call from the API 25, as indicated by step (1) of Fig. 2A. At step (2) the function call is subsequently passed from the request router 30 to the notification manager 32.

In turn, the notification manager 32 sequentially notifies each of the extensions 34, 35, at steps (3) and (4), respectively, of the pending call before the call is made (pre-notification). This pre-notification step is also shown in Fig. 2B as step 16.

During this pre-notification procedure the extensions 34, 35 are called, enabling the extensions to execute their own procedures and perform operations according to their extended functionality thereby providing extensibility to the pending operation request. The extensions 34, 35 are, however, limited in how they can ultimately affect the subsequent passing of the pending request to the appropriate service provider.

Following pre-notification of the extensions, the notification manager returns control of the call to the request router 30 at step (5), which then routes the call to one or more appropriate service providers, such as the service provider 28, at step (6). The service provider 28 services the requesting call, and when completed, returns control to the request router 30 at step (7). The call or calls to the service providers are also shown in Fig. 2B as step 18.

After the operation request has been passed to the service provider 28 and handled thereby, at step (8) of Fig. 2A the request router 30 again transfers control of the call to the notification manager 32. At steps (9) and (10) the notification manager 32 notifies the extensions 34, 35 of the results of the call, (post-notification), giving the extensions 34, 35 an opportunity to again provide extensibility to the function call.

During the post-notification, the extensions may, for example, influence the results returned from the called function, request a retry of a failed call, or execute their own procedures. By way of example, one of the extensions 34 may alter a result returned by the service provider indicative of failure, to fool the application program into believing that the call was successfully completed. Post-notification of the extensions is also shown in Fig. 2B as step 19.

Once the post-notification procedure is completed, the notification manager 32 returns control to the request router 30 at step (11). Finally, the request router 30 returns a result to the application program 23 at step (12) of Fig. 2A, (which corresponds to step 22 of Fig. 2B).

<u>The Cheng Reference</u>

Cheng is directed to computer-implemented database systems, and, in particular, to an extender for a computer-implemented relational database system for storing, querying, and retrieving structured documents. Cheng also relates to an extender for a computer-implemented relational database system, including indexing of structured documents with general and rich data types, querying structured documents using a novel conditional select function; and creating structure indexes using a novel tag counting system.

As noted in Cheng, relational extenders are entities created to help relational database users handle complex data types. Relational extenders define and implement complex data types, storing the attributes, structure, and behavior of the data types in a column of a relational database table. The complex data types stored in relational databases support new applications to be run and/or extend existing business applications. Within the relational database system, these data types need to be manipulated through the standard Structured Query Language (SQL). As a result, relational extenders provide management solutions for handling different types of data.

Cheng further describes an XML extender for a computer-implemented relational database system for storing, querying, and retrieving structured documents. Generally, relational extenders define and implement complex data types and extend the tables within the relational database with the new data types. The XML extender provides an Abstract Data Type (ADT) DB2XML, which can be specified as a column data type, and includes several User Defined Functions (UDFs) for storing, searching, and retrieving XML documents internally, as DB2.RTM. Character Based Large Objects (CLOB), or externally, in flat files or Uniform Resource Locators (URLs), for example.

Applicant's Disclosure

Applicant's disclosure is directed to providing new ***software delivery models*** that are particularly well-suited for network-based software delivery, e.g. delivery via the Internet. See, Specification, page 2, lines 1-22.

Neither Smale nor Cheng are directed to providing methods and systems for ***delivering*** software, as contemplated in Applicant's disclosure. Rather, Smale is directed to providing methods and systems that function, in connection with software extensions, only once the extensions are already registered and present on the system. Cheng is directed to a relational extender for use in connection with a relational database. Cheng is not directed to methods and systems for delivering software, as contemplated in Applicant's disclosure. Once these fundamental differences are appreciated, the patentable distinctions between Applicant's claimed subject matter and the cited reference is more easily appreciated.

As noted in Applicant's "Overview" section starting on page 6, line 1, Applicant's methods and systems provide a mechanism by which functionality can be added dynamically to an application program or software platform. Functionalities or extensions can be advantageously added via a network such as the Internet. Extensions, that can implement new features or add to existing features, can be added using only a network address, such as a URL, as a basis for extension installation. That is, all of the files that comprise an extension can be maintained on the network and accessed via one or more network sites.

Extensions can be described in a variety of ways. One way utilizes a hierarchical tag-based language which facilitates handling and use of the various extensions. In one particular implementation, a software platform is provided that can incorporate various functionalities. The software platform and the inventive architecture enable third and fourth party developers to develop extensions for the platform that can be easily and seamlessly incorporated into the platform without having any knowledge of (or relationship with) a hosting service. A third party developer is a developer who develops an extension for the platform. A fourth party developer might be a developer who develops an extension to a third party developer's extension. Thus, the incorporation of third and fourth party extensions is essentially a transparent process, as far as developers are concerned.

Consider for example, Applicant's Fig. 1, which shows a user's computer 100 and several so-called extension sources 102, 104, and 106. The extension sources can comprise any entity from which a software extension can be obtained via a network. In an exemplary implementation, the network can comprise the Internet, although other networks (e.g. LANs and WANs) can certainly be utilized. Extension sources can include, without limitation, business entities such as retail stores that might maintain a network site. In one implementation, a user can execute software on their computer that provides an application program or software platform. Each of the different extension sources 102-106 can provide software extensions that can plug into the software platform that is executing on the user's machine. These extensions are deliverable via a network such as the Internet, and assist in providing applications that can be executed on the user's machine. In some embodiments, the extensions are logically described in XML. Additionally, the use of XML assists in the future discoverability of extensions by promoting XML DOM properties on the Internet. It will be appreciated, however, that any suitable format can be used for describing the extensions, e.g. a binary description could be used.

Extensions can be delivered from any number of different extension sources. The inventive methods and systems provide a streamlined and organized way to handle the provided extensions. The use of XML advantageously enables efficient handling of extensions from multiple different extension sources, without unduly taxing the software components that utilize specific portions of an extension or extensions.

One exemplary system that can embody features of Applicant's inventive systems and methods is shown and described in connection with Fig. 17.

In one described embodiment, one of the aspects that provide desirable utility is the extensibility of the software platform. That is, third and fourth party developers are free to develop their own extensions which can then be used within the framework of the software platform. The extensions are integrated directly into the software so that the platform's functionality is modified by the extensions. To provide an extension, a software developer simply authors the extension, describes their extension in an Extension Definition File (EDF) and a Package

Manifest (PKG), and then hosts the EDF, PKG, and associated files on a network server.

The EDF, as pointed out the Specification, can be defined in an XML schema that includes a root node (i.e. the "extension" tag) and one or more child nodes. In this particular example, the child nodes generally correspond to the individual extension feature types that are desired for incorporation into the software platform. Tables 1-3, in the Specification, describe various exemplary predefined feature types that can be added through an extension using the predefined XML schema.

Consider now a developer who wants to add two menus and a toolbar to the software platform. The menus and toolbar might be associated with a retail store that maintains a Web site for its customers. The retail store might want a customer who visits its Web site to be presented with a UI that is unique to the retail store and provides services that are specifically tailored to the store. To do this, the developer develops two different menus, one of which might be associated with displaying the most recent specials, and other of which might be associated with providing a search mechanism through which the user can search for specific products. The toolbar might contain specific buttons that are unique to the retail store. A simplified EDF called "*retail.edf*" for the retail store's extension is shown directly below:

```
<edf:extension  name=  "retail extension" urn= "extension.retail.com">
        <edf:menus>
                <edf:menu    url= "url1.htm"/>
                <edf:menu    url= "url2.htm"/>
        </edf:menus>
        <edf:toolbars>
```

```
            <edf:toolbar url= "url3.htm"/>
        </edf:toolbars>
    </edf:/extension>
```

Here, the outer "extension" tag designates this XML file as an extension. The inner "menus" and "toolbars" tags are top level tags that designate that the information between these tags pertains respectively to menus and toolbars that correspond to the extensions that the developer has added. The boldface "menu" and "toolbar" tags describe data pertaining to the actual extension and contain a URL that is associated with each extension. The EDF above logically describes the extensions that are being provided as including two menus and one tool bar.

Consider also that the above EDF can constitute but one of many EDFs that are loaded into the system. Each EDF can contain one or more top level tags, each of which is associated with one or more specific extensions that are to be added to the software platform.

Fig. 17 is a block diagram of an exemplary software architecture that is configured to process multiple different EDFs so that the software components that are responsible for incorporating each particular extension into the software platform receive the appropriate information that is specific to their extension. This example is specific to the XML implementation that is discussed throughout the Specification.

Utility objects, herein referred to as "attachment points", are used to process the information from the multiple EDFs. An attachment point is simply a collection of objects that fire events to registered listeners as objects are added to or removed from the collection. Many types of attachment points can be created, but all take data from a source (often another attachment point), process it (either dynamically or statically), and expose the results of their processing. Some of the simplest attachment points include:

- An XML attachment point, which loads an XML file and exposes the top-level nodes of the XML as objects in its collection.
- A filter attachment point, that connects to another attachment point and exposes only those objects from it that meet some criteria.
- A merge attachment point, that connects to one or more other attachment points and exposes all of their objects as one, merged collection of objects.

In the illustrated example, the architecture includes a collection of one or more attachment points, including a funnel structure known as an EDFHub 1700, an attachment point manager 1702, and multiple attachment managers 1704. The EDFHub 1700 receives all of the EDFs and merges them together and exposes them as a single list. Other individual attachment points provide mechanisms that manipulate (including filter, merge and expand) the single list that is exposed by the EDFHub 1700.

Whenever a new extension or EDF is added to or removed from the EDFHub, the various attachment points will see to it that the appropriate attachment manager(s) is notified. This is done by firing events to the appropriate attachment managers. The attachment point manager 1702 creates, destroys and manages the various attachment points in the system and allows easy reuse of identical attachment points.

For each top level tag (i.e. "menus" and "toolbars" tags), there is a corresponding attachment manager 1704 that uses data provided by the attachment points to incorporate a particular type of feature within the software platform. Each attachment manager requests a set of attachment points from the attachment point manager 1702. These manipulate the data exposed by the EDFHub 1700. In the illustrated example, the attachment points can be requested as a predicate chain that the attachment point manager uses to create and build a set of attachment points that operate on the data exposed by the EDFHub 1700.

Attachment points and the Attachment Point Manager are described in more detail in the Specification, starting at page 45, line 10.

The Claim Rejections

Claim 1 recites a software architecture embodied on a computer-readable medium, the architecture comprising:

- multiple attachment points collectively arranged to filter data associated with ***files that describe software extensions***; and
- multiple extension managers ***associated with the multiple attachment points and with respective feature types that can be added to a software platform by software extensions***, the extension managers being configured to receive data from the multiple attachment points that pertains only to the feature type with which the extension manager is associated.

In making out the rejection of this claim, the Office cites to Smale, column 2, lines 39-42, and column 3, lines 1-10, apparently for the proposition that this excerpt discloses "multiple attachment points" as recited in this claim; and to column 3, lines 1-15 apparently for the proposition that this excerpt discloses "multiple extension managers" as recited in this claim.

However, these excerpts simply discuss Smale's methods and systems as they pertain to coordinating software extensions to eliminate perceived deficiencies. In addition, these excepts describe the general construction of

Smale's system in which registered extensions operate in connection with a central manager which intercepts calls from an application for the purpose of pre-notifying and post-notifying the extensions as described above, to allow the extensions to impart their functionality to the system.

Nowhere does Smale disclose or suggest multiple attachment points collectively arranged to filter data associated with ***files that describe software extensions***. Further, nowhere does Smale disclose or suggest multiple extension managers ***associated with the multiple attachment points and with respective feature types that can be added to a software platform by software extensions.***

Accordingly, for each of these individual reasons, claim 1 is not anticipated by Smale and is hence allowable.

Claims 2-8 depend from claim 1 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 1, are neither disclosed nor suggested in the references of record, either singly or in combination with one another. In addition, given the allowability of these claims, the Office's reliance on Cheng adds nothing of significance. Further, the Office has failed to make out a prima facie case of obviousness for at least the reasons that the Office's stated motivation for combining Smale and Cheng is conclusory, is based on hindsight reconstruction, and is not stated with particularity, as it must be.

Claim 9 recites a software architecture embodied on a computer-readable medium comprising:

a hub structure configured to:

- ***receive multiple different files that describe software extensions*** that can be added to a software platform;

- *combine the multiple different files* into a single exposable list; and
- *expose the single exposable list to a filter structure* that is configured to filter the list.

In making out the rejection of this claim, the Office cites to Smale's column 2, lines 39-42, column 3, lines 1-15 and Fig. 3, as apparently disclosing a hub structure configured to receive multiple different files that describe extensions that can be added to a software platform. Applicant respectfully disagrees. The excerpt cited by the Office describes Smale's system which simply includes a central manager that intercepts calls from an application program and re-directs the calls to individual extensions in a pre-notification and post-notification process. This in no way discloses or suggests a hub structure configured to ***receive multiple different files that describe software extensions*** that can be added to a software platform. Accordingly, for at least this reason, this claim is allowable.

Further, the Office cites to Smale's column 3, lines 38-40, column 7, lines 24-43 and Figs. 4 and 5, as apparently disclosing a hub structure that combines the multiple different files into a single exposable list, and exposes the single exposable list to a filter structure that is configured to filter the list. Applicant respectfully disagrees.

The excerpt cited by the Office simply discloses a data structure that is utilized by Smale to contain information relating to extensions that request notification when a particular type of call is received by the central manager. Thus, the data structure is simply one which is used to organize and re-direct calls to the various extensions that are registered in Smale's system. This in no way discloses or suggests a hub structure that ***combines the multiple different files*** into a ***single exposable list***, and ***exposes the single exposable list to a filter structure*** that is configured to filter the list. Accordingly, for at least this additional reason, this claim is allowable.

Claims 10-11 depend from claim 9 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 9, are neither disclosed nor suggested in the references of record, either singly or in combination with one another. In addition, given the allowability of claim 9, the Office's reliance on Cheng to reject claim 10 does not add anything of significance. Further, the Office has failed to make out a prima facie case of obviousness for at least the reasons that the Office's stated motivation for combining Smale and Cheng is conclusory, is based on hindsight reconstruction, and is not stated with particularity, as it must be.

Claim 12 recites a software architecture embodied on a computer-readable medium, the architecture comprising multiple different attachment points each of which is configured to:

- receive XML data that pertains to one or more software extensions that can be added to a software platform;
- process the XML data to provide a list of XML nodes; and
- expose the list of XML nodes.

In making out the rejection of this claim, the Office argues that Smale teaches a software architecture comprising multiple attachment points each of which is configured to receive data as recited in this claim. Applicant disagrees.

First, Smale does not teach or suggest attachments points as that term is contemplated in Applicant's disclosure.

Further, this claim recites that each of the attachment points is configured to, *inter alia*, receive XML data that pertains to one or more software extensions ***that can be added*** to a software platform. As Smale's extensions are already registered in the system, Smale in no way teaches an architecture in which individual attachment points are configured to receive any such data that pertains to one or more software extensions that ***can be added*** to a software platform. In as much as Smale is missing this feature completely, it is virtually impossible for Smale to disclose the features that follow in the claim and which depend on this first feature.

The Office further admits that Smale does not teach XML data, providing a list of XML nodes, and exposing a list of XML nodes. Applicant agrees. The Office then relies on Cheng's XML disclosure and argues that it would be obvious to combine the teachings of both of these references to render the subject matter of this claim obvious. As a stated motivation for making the combination, the Office argues that "[t]he modification would be obvious because one of ordinary skill in the art would be motivated to provide extensions using an extensible programming language and to provide extensions to World Wide Web documents."

The Office has failed to make out a prima facie case of obviousness for a number of different reasons. First, as noted above, all of the features recited in this claim are not disclosed or taught in the cited references. These absences cannot be provided by assuming that a reference discloses that which it does not. Second, the Office's stated motivation for making the combination is, at best, conclusory and based on hindsight reconstruction that is not supported with the particularity that the Federal Circuit says it must be. The Office's stated motivation is equivalent to saying "It would be obvious to create this invention because it would be obvious to create this invention." Accordingly, for at least these additional reasons, this claim is allowable.

Claims 13-17 depend from claim 12 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 12, are neither disclosed nor suggested in the references of record, either singly or in combination with one another.

Claim 18 recites a software architecture embodied on a computer-readable medium, the architecture comprising:

- a hub structure configured to:
  - receive multiple different files that describe software extensions that can be added to a software platform;
  - combine the multiple different files into a single exposable list; and
  - expose the single exposable list to a filter structure that is configured to filter the list;
- a filter structure comprising multiple attachment points collectively arranged to filter data associated with the list exposed by the hub structure; and
- multiple extension managers associated with the multiple attachment points and with respective feature types that can be added to a software platform by software extensions, the extension managers being configured to receive data from the multiple attachment points that pertains only to the feature type with which the extension manager is associated.

This claim recites features which appear in claim 9's recited hub structure. Thus, for all of the reasons set forth with respect to the allowability of claim 9, this claim is allowable. In addition, this claim recites features (i.e. a filter structure, and multiple extension managers) which, in combination with the recited hub structure, are simply not disclosed in Smale.

Claims 19-25 depend from claim 18 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 18, are neither disclosed nor suggested in the references of record, either singly or in combination with one another. Given the allowability of claim 18, the Office's reliance on Cheng to reject claims 19-24 is not seen to add anything of significance. Further, the Office has failed to make out a prima facie case of obviousness for at least the reasons that the Office's stated motivation for combining Smale and Cheng is conclusory, is based on hindsight reconstruction, and is not stated with particularity, as it must be.

Claim 26 recites a method of ***providing a software extension*** comprising:

- exposing an XML list that contains one or more nodes;
- processing the XML list to identify specific nodes that correspond to various feature types that can be added to a software platform; and
- notifying an extension manager that is associated with at least one feature type if a node that corresponds to that feature type is identified in the XML list.

In making out the rejection of this claim, the Office argues that this claim is simply a method version of the software architecture of claim 12. As such, the Office rejects this claim for the same reasons as claim 12. Applicant disagrees with the Office and submits that this claim is not simply a method version of claim 12. Specifically, this claim recites features which are different from those recited in claim 12. For example, this claim recites "notifying an extension manager that is associated with at least one feature type if a node that corresponds to that feature type is identified in the XML list." Claim 12 contains no such subject matter. Applicant submits that this claim has not been examined by the Office because the cited art has not been applied to this claim.

Assuming, however, that the Office had specifically applied the art to this claim, it is evident, in light of the discussion above, that neither Smale nor Cheng, individually or in combination with one another, disclose or suggest the subject matter of this claim. Accordingly, for at least this reason, this claim is allowable.

Claims 27-32 depend from claim 26 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 26, are neither disclosed nor suggested in the references of record, either singly or in combination with one another.

Claim 33 recites a method of ***providing a software extension*** comprising:

- receiving XML data that pertains to a software extension that is to be added to a software platform;
- processing the XML data to identify XML nodes; and
- exposing an XML list that contains one or more nodes that are identified by said processing.

In making out the rejection of this claim, the Office argues that this claim is simply a method version of the architecture recited in claims 18-19. Accordingly, the Office rejects this claim for the same reasons as claims 18-19 were rejected. Applicant disagrees with the Office and submits that this claim is not simply a method version of claims 18-19. Specifically, this claim recites features which are different from those recited in claims 18-19. Applicant submits that this claim has not been examined by the Office because the cited art has not been applied to this claim.

Assuming, however, that the Office had specifically applied the art to this claim, it is evident, in light of the discussion above, that neither Smale nor Cheng, individually or in combination with one another, disclose or suggest the subject matter of this claim. Specifically, neither Smale nor Cheng disclose or suggest a method that "receives XML data that pertains to a software extension that ***is to be added*** to a software platform." Accordingly, for at least this reason, this claim is allowable.

Claims 34-39 depend from claim 33 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 33, are neither disclosed nor suggested in the references of record, either singly or in combination with one another.

Claim 40 recites a method of ***providing a software extension*** comprising:

- receiving multiple different files, each of which being associated with a different software extension and logically describing its associated software extension;
- combining the multiple different files in a single list;
- exposing portions of the list;
- processing exposed portions of the list to identify one or more feature types that are to be added to a software platform; and
- notifying an extension manager that is associated with a particular feature type.

In making out the rejection of this claim, the Office argues that this claim is a method version of the architecture recited in claim 18. Applicant respectfully disagrees. Specifically, this claim recites features which are different from those recited in claim 18. Applicant submits that this claim has not been examined by the Office because the cited art has not been applied to this claim.

Assuming, however, that the Office had specifically applied the art to this claim, it is evident, in light of the discussion above, that neither Smale nor Cheng, individually or in combination with one another, disclose or suggest the subject matter of this claim. Specifically, neither Smale nor Cheng disclose or suggest a method that "receives multiple different files, each of which being associated with a different software extension and logically describing its associated software extension." Accordingly, for at least this reason, this claim is allowable.

Claims 41-42 depend from claim 40 and are allowable as depending from an allowable base claim. These claims are also allowable for their own recited features which, in combination with those recited in claim 40, are neither disclosed nor suggested in the references of record, either singly or in combination with one another.

Conclusion

All of the claims are in condition for allowance. Accordingly, Applicant requests a Notice of Allowability be issued forthwith. If the Office's next anticipated action is to be anything other than issuance of a Notice of Allowability, Applicant respectfully requests a telephone call for the purpose of scheduling an interview.

Amended Specification with Markups to Shows Amendments

In the Specification, please replace the text appearing on page 1, line 1 through line 19 with the following:

-- **RELATED APPLICATIONS**

The following patent applications are related to the present application, are assigned to the assignee of this patent application, and are expressly incorporated by reference herein:

- U.S. Patent Application Serial No. 09/599,298, entitled "Single Window Navigation Methods and Systems", bearing attorney docket number MS1-560us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. 09/599,806, entitled "Methods and Systems of Providing Information to Computer Users", bearing attorney docket number MS1-557us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. 09/599,299, entitled "Methods, Systems, Architectures and Data Structures For Delivering Software via a Network", bearing attorney docket number MS1-559us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. 09/599,048, entitled "Network-based Software Extensions", bearing attorney docket number MS1-563us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. 09/599,813, entitled "Authoring Arbitrary XML Documents Using DHTML and XSLT", bearing attorney docket number MS1-583us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. 09/599,086, entitled "Task Sensitive Methods And Systems For Displaying Command Sets", bearing attorney docket number MS1-562us, and filed on the same date as this patent application.--

<u>Amended Claims with Markups to Shows Amendments</u>

1. (Amended) A software architecture <u>embodied on a computer-readable medium, the architecture</u> comprising:

multiple attachment points collectively arranged to filter data associated with files that describe software extensions; and

multiple extension managers associated with the multiple attachment points and with respective feature types that can be added to a software platform by software extensions, the extension managers being configured to receive data from the multiple attachment points that pertains only to the feature type with which the extension manager is associated.

9. (Amended) A software architecture <u>embodied on a computer-readable medium, the architecture</u> comprising:

a hub structure configured to:

receive multiple different files that describe software extensions that can be added to a software platform;

combine the multiple different files into a single exposable list; and

expose the single exposable list to a filter structure that is configured to filter the list.

12. (Amended) A software architecture <u>embodied on a computer-readable medium, the architecture</u> comprising multiple different attachment points each of which is configured to:

receive XML data that pertains to one or more software extensions that can be added to a software platform;

process the XML data to provide a list of XML nodes; and

expose the list of XML nodes.

18. (Amended) A software architecture embodied on a computer-readable medium, the architecture comprising:

a hub structure configured to:

receive multiple different files that describe software extensions that can be added to a software platform;

combine the multiple different files into a single exposable list; and

expose the single exposable list to a filter structure that is configured to filter the list;

a filter structure comprising multiple attachment points collectively arranged to filter data associated with the list exposed by the hub structure; and

multiple extension managers associated with the multiple attachment points and with respective feature types that can be added to a software platform by software extensions, the extension managers being configured to receive data from the multiple attachment points that pertains only to the feature type with which the extension manager is associated.

23. (Amended) The software architecture of claim 19, wherein an extension manager is notified whenever [a] an extension comprising a feature type with which it is associated is added or removed from the software platform.

30. (Amended) The method of claim 26, wherein said processing is accomplished by [exploding] <u>exposing</u> various nodes.

31. (Amended) The method of claim 26, wherein said processing is accomplished by filtering on specific nodes and [exploding] <u>exposing</u> various nodes.

37. (Amended) The method of claim 36, wherein at least one of the attachment points [expodes] <u>exposes</u> a node.

Respectfully Submitted,

Dated: 5/29/03

By: [illegible]
Lance R. Sadler
Reg. No. 38,605
(509) 324-9256